A Hybrid Programming Model for Compressible Gas Dynamics using OpenCL

Ben Bergen
Applied Computer Science (CCS-7)
Los Alamos National Laboratory

Marcus Daniels
Applied Computer Science (CCS-7)

Paul Weber
High-Performance Systems Integration (HPC-5)
Why are programming models important?

The growing variety of heterogeneous and multicore HPC computing architectures means that we need new tools and programming models if we are to sustain code portability.

- **Roadrunner**
  - AMD + IBM Cell

- **NCSA Blue Waters**
  - Power7 + Accelerator

- **Cray Heterogeneous**
  - AMD + nVIDIA

- **Sequoia**
  - Blue Gene Q

- **Desktop**
  - Multicore + Accelerator
Some Challenges for Radiation Hydrodynamics on Modern Computing Architectures

- **Portability (#4)**
  - Current architectural diversity is challenging to support with single source base
  - Optimization strategies often depend on underlying architecture

- **Concurrency (#3)**
  - Hierarchy of parallelism: instruction-level, data, task, distributed-memory
  - Need expressions of radhydro algorithms to address all of them
  - Difficult to increase amount of data-parallelism
    - Affects scalability (think Opteron-only use of Roadrunner)
  - Task-level parallelism offers some promise

- **Fault Tolerance (#2)**
  - MTBF could be less than the time it takes to write a checkpoint

- **Data Motion (#1)**
  - Scalability will primarily be limited by power consumption
  - Data movement = Power
Hierarchy of Parallelism

Node → Node → Node → Node

Node

Architecture A

Core

Stg 0
Stg 1
Stg 2
Stg 3

FPU

Architecture B

Distributed-Memory
Task
Data + Task
Superscalar
Instruction-Level
SIMD
Future systems will require hierarchical programming models for scalability, fault tolerance and power efficiency.
One Possible Model

Use different tools for inter and intra-node control

- MPI
  - Distributed-Memory
  - Task
  - Data + Task
  - Superscalar
  - Instruction-Level
  - SIMD

- OpenCL
What is OpenCL?

OpenCL is a framework for applications development on multicore, manycore and accelerated architectures

- **Runtime Environment**
  - Work distribution, dynamic kernel compilation

- **Application Programming Interface (API)**
  - Process launch, communication, synchronization
  - Topology interrogation (resource detection)

- **OpenCL C Kernel Language**
  - Low-level computational kernel language
  - Subset of C with extensions
  - Abstract vector types and intrinsics

```c
for(i=0; i<N; i++) {
    rho(i) = rho(i) + C*(rFR – rFL);
    rhou(i) = rhou(i) + C*(ruFR - ruFL);
} // for
```
Proof-of-Concept Application

- **High-resolution direct Eulerian hydrodynamics solver**
  - MUSCL-Hancock Godunov method
  - Solves two-dimensional Euler equations
  - Structured grid with reflecting boundaries
  - Surface plot shows density on z-axis

- **Distributed-memory parallel with MPI**

- **Implemented with C++ and OpenCL**
  - Single-source implementation runs on all architectures
  - SC09 Demo ran on five different architectures using OpenCL compilers from three different vendors

AMD Multicore | Intel Multicore | IBM Multicore (P6) | IBM Cell | nVIDIA GPU
BladeCenter H System at SC09 (contributed by IBM)
Exploiting Hybrid Architectures (Inter-Node)

- **Demo Code: Synchronous model**
  - Control rank initiates all compute tasks
  - Not a viable model: will not scale

- **Enhancement: Asynchronous model**
  - Compute ranks accept requests from control rank, but perform updates without external initiation
Exploiting Hybrid Architectures (Intra-Node)

Host Process/Task Queue

Device

Scheduler

Disk I/O

Computation

\[ F_{net} = \frac{d(mv)}{dt} = m \frac{dv}{dt} \]

Device

Diagnostics/
Rendering

LA-UR 10-04896
### Compute Process Event Structure (Compute Ranks)

**Host**  
MPI + OpenCL

---

**Event Handler**

- **Test For Completion (simulation)**
- **Advance Simulation**
- **Execute Dynamic Events**
- **Execute Scheduled Events**
- **Check For Control Request**

---

**Performance Queue**

---

**Auxiliary Queue**

---

**Auxiliary Queue**

---

Operated by Los Alamos National Security, LLC for NNSA
Functional Programming Model

- An OpenCL NDRange can be thought of as applying a functional over an index space
  \[ f \forall a f(i) \forall i \in I \]

- The algorithm must be data-parallel, i.e., OpenCL functional model cannot honor lexicographical dependencies

- Standard data-parallel nested for-loop is a special case where the index is determined by a mapping from logical \( N \)-dimensional space to the index space

\[ i \in (M_x(k, j) \forall k \in [0K N_y - 1], j \in [0K N_x - 1]) \]

```c
for(k=0; k<N_y; ++k) {
    for(j=0; j<N_x; ++j) {
        c[k][j] = a[k][j] + b[k][j];
    } // for
} // for
```
Functional Expression of MUSCL-Hancock

\[ u_i = \frac{\rho u_i}{\rho_i} \]

\[ \sigma_i = \omega (q_{i+1} - q_i) \]

\[ \rho_{Tail_i} = \rho_i + \frac{\Delta x}{2} \sigma_i \]

\[ f_i^\rho = \rho u_{Tail_i} - \rho u_{Head_i} \]

\[ \hat{\rho}_i = \rho_i - \zeta f_i^\rho \]
Kernel Fusion

- **Original work used by-hand fusion**
  - Ugly, tedious and error-prone
  - Excellent performance increases

- **Kernel expression and optimization for functional programming**
  - GRA summer project (Ian Karlin will continue this work as a postdoc)
  - Compiler used to fuse kernels to limit data motion and increase arithmetic intensity
    - Project used POCC and Pluto to perform optimizations, e.g., array contraction

Unfused

- Input
- Temporary
- Temporary
- Output
- Update A
- Update B
- Update C

Fused

- Input
- Update A
- Update B
- Update C
- Output
MUSCL-Hancock Numerical Support

Support = 2

Step 1

Step 2

Steps 3-4

Step 5

Steps 6-8

Step 9
Kernel Fusion Improvement: Magny-Cours

![Graph showing performance improvement with different grid sizes for Opteron 6136 processor. The graph compares unfused, partially fused, and fully fused states.](image)
Conclusions and Future Work

- Hybrid Programming Model
  - Viable start for Exascale
  - Leverages existing high-level code structure
  - Utilizes existing tools

- Kernel Fusion
  - Good strategy on all tested architectures
  - CPU performance is better for small problem sizes
    - Intuitive result: GPU data must traverse PCIe bus
  - Better version leaves data on the device
    - Only nearest-neighbor data moved off device

Heterogeneous designs like the IBM Cell and AMD Fusion offer the best of both worlds. Future architectures will likely converge to this paradigm.